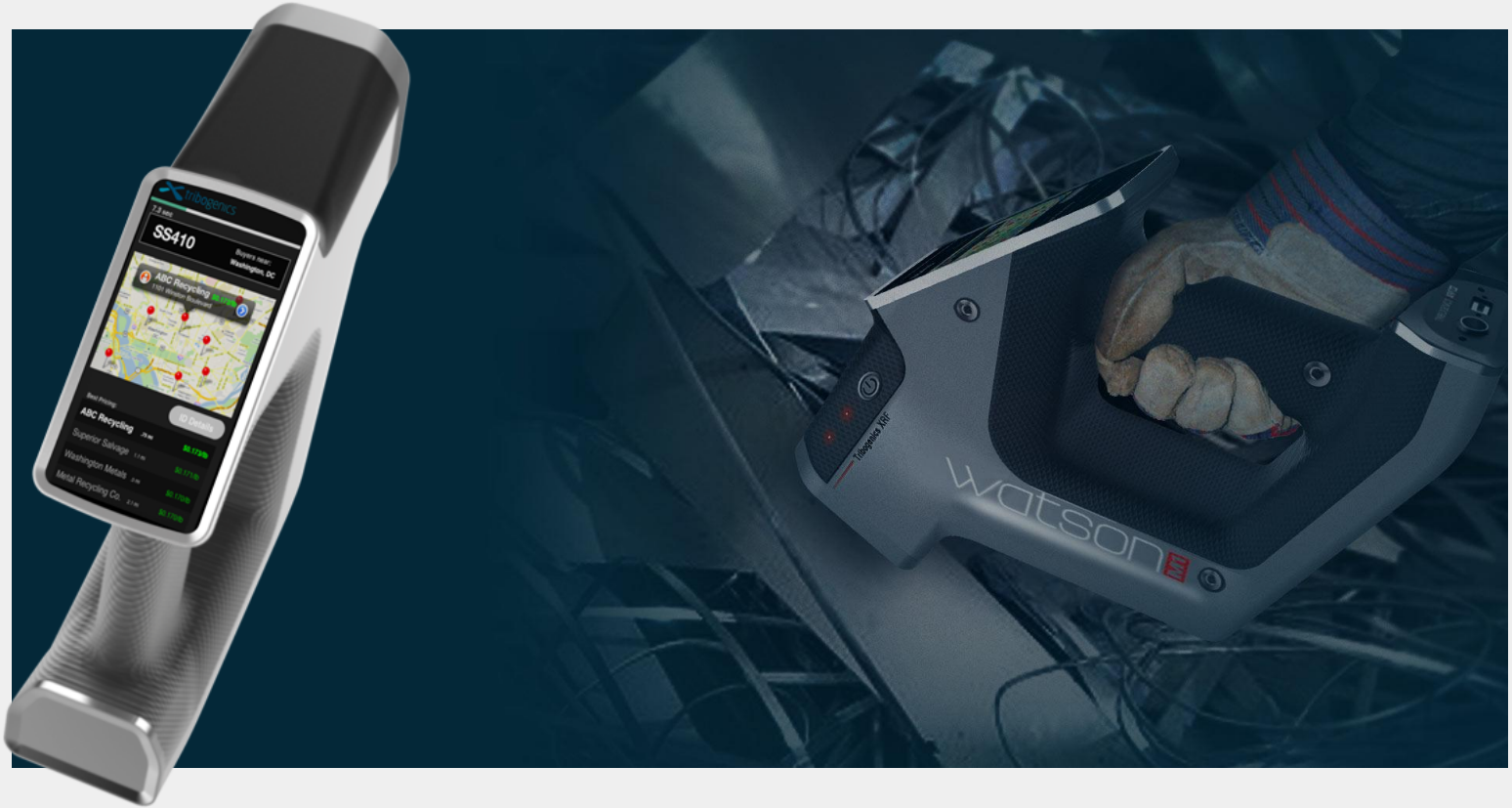
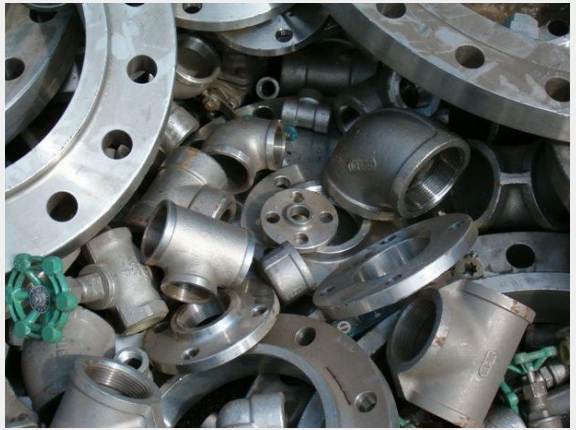
Utilizing the Android Open Source Project to Support Controllers for Single-Use Devices

X-Ray Guns! Pew Pew!

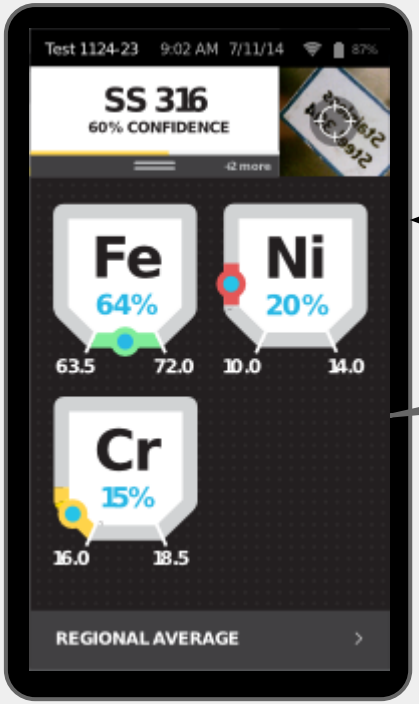
Watson Project Overview



Watson Project Overview



Watson Architecture



Watson Controller Board

X-Ray Source

External Battery

USB Peripherals

X-Ray Detector

Camera

LEDs

Trigger

What we’re covering

What we’re covering

- AOSP Customizations

What we’re covering

- AOSP Customizations
- App-specific Features

What we’re covering

- AOSP Customizations
- App-specific Features
- Problems that we overcame

AOSP Customization

AOSP Customization

SDG SYSTEMS™

- Pre-5.x Kiosk Mode

AOSP Customization

- Pre-5.x Kiosk Mode
- Altering the status bar

AOSP Customization

- Pre-5.x Kiosk Mode
- Altering the status bar
- Using the Android settings GUI in our application

AOSP Customizations

SDG SYSTEMS™

Where to get it

https://source.android.com/source/

Basic file structure

frameworks/base - Android-specific code

packages/apps - Where to put your app / apk

device/ [lge/hammerhead] - hw-specific config files

AOSP Customizations - Build specifics


```
export USE_CCACHE=1
export LANG=C
export PATH=$PATH:/usr/lib/jvm/java-1.7.0-openjdk-amd64/bin

#Set an awesome (and transient) output directory
export OUT_DIR_COMMON_BASE=/home/volatile/bfriedberg

. build/envsetup.sh
lunch
<pick your release, I used 'aosp_hammerhead-userdebug'>
make [-j30]
```

Kiosk mode

Kiosk mode

There are many options

- Car mode
- Launcher replacement
- Pinned Activities (5.0+)
- Sealed Mode (5.0+)

http://sdgsystems.com/blog/implementing-kiosk-mode-android-part-1/

Kiosk mode

Android navigation paradigms get in the way

- Home button
- Back button
- Notification Bar

Kiosk mode

Our approach

- Launcher Removal / Replacement
- Fake Hardware Keys (to remove soft buttons)
- Override the notification bar

Kiosk mode - Launcher Replacement

SDG SYSTEMS™

Add your app to packages/apps with this Android.mk file:

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := <package_name>
LOCAL_MODULE_CLASS := APPS
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
LOCAL_CERTIFICATE := platform
LOCAL_OVERRIDES_PACKAGES := Launcher2 Launcher3 #this is optional
LOCAL_SRC_FILES := <apk_name>.apk
LOCAL_MODULE_OWNER := system
include $(BUILD_PREBUILT)
```

Kiosk mode - Launcher Replacement

In device/lge/hammerhead/aosp_hammerhead.mk:

Delete Launcher3 and add the name of your app to the PRODUCT_PACKAGES collection

Kiosk mode - Launcher Replacement

In your App's AndroidManifest.xml:

```
<activity
     android:name=".ui.SplashActivity"
     android:screenOrientation="portrait"
     android:windowSoftInputMode="stateAlwaysVisible" >
     <intent-filter android:priority="1000" >
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
          <category android:name="android.intent.category.HOME" />
          <category android:name="RUN_AT_BOOT" />
          <category android:name="android.intent.category.DEFAULT" />
     </intent-filter>
</activity>
```

Kiosk mode - Fake Hardware Keys

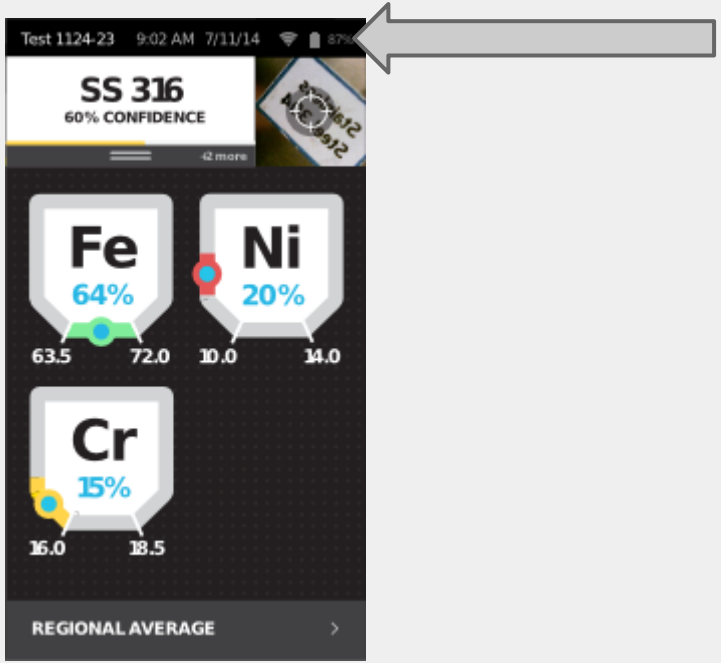
In device/lge/hammerhead/aosp_hammerhead.mk

```
# Disable the navigation bar
PRODUCT_PROPERTY_OVERRIDES += \
    qemu.hw.mainkeys=1
```

Kiosk mode - Notification Bar

```
public void disableNotificationBarShade() throws ClassNotFoundException {
     Object object = this.getSystemService("statusbar");
     Class<?> statusBarManager;
     statusBarManager = Class.forName("android.app.StatusBarManager");
     Method[] method = statusBarManager.getMethods();
     Field fld[] = statusBarManager.getDeclaredFields();
     Class name = object.getClass();
     Field field = statusBarManager.getDeclaredField("DISABLE_EXPAND");
     Method disable = statusBarManager.getMethod("disable", new Class[] { int.class });
     disable.setAccessible(true);
     field.setAccessible(true);
     disable.invoke(object,Integer.valueOf(field.getInt(statusBarManager)));
}
```

Altering the status bar



Altering the status bar

- Battery level

Altering the status bar

- Battery level
- Add labels

Altering the status bar

- Battery level
- Add labels
- Change Icons

Altering the status bar - Battery level

frameworks/base/packages/SystemUI/src/com/android/systemui/BatteryMeterView.java:

```
public class BatteryMeterView extends View implements DemoMode {
…
//public static final String mBatteryIntent = Intent.ACTION_BATTERY_CHANGED;
public static final String mBatteryIntent = "<package>.action.battery_changed";
…
// if (action.equals(Intent.ACTION_BATTERY_CHANGED)) {
if (action.equals(mBatteryIntent)) {
<etc…>
```

In your app, issue the intent it with:

```
intent.putExtra(BatteryManager.EXTRA_LEVEL, level); //0-100
intent.putExtra(BatteryManager.EXTRA_PLUGGED, chargingValue); //[0,1]
sendBroadcastAsUser(intent, new UserHandle(Parcel.obtain()));
```

Altering the status bar - Battery level

frameworks/base/packages/SystemUI/src/com/android/systemui/statusbar/policy/BatteryController.java -- Change the constructor to take a custom intent action

frameworks/base/packages/SystemUI/src/com/android/systemui/statusbar/phone/PhoneStatusBar.java -- Construct the mBatteryController with the custom intent

Altering the status bar - Adding labels

Status bar layout is in

`frameworks/base/packages/SystemUI/res/layout/status_bar.xml`

Add a new TextView (or whatever) and then use a custom intent to manipulate it in

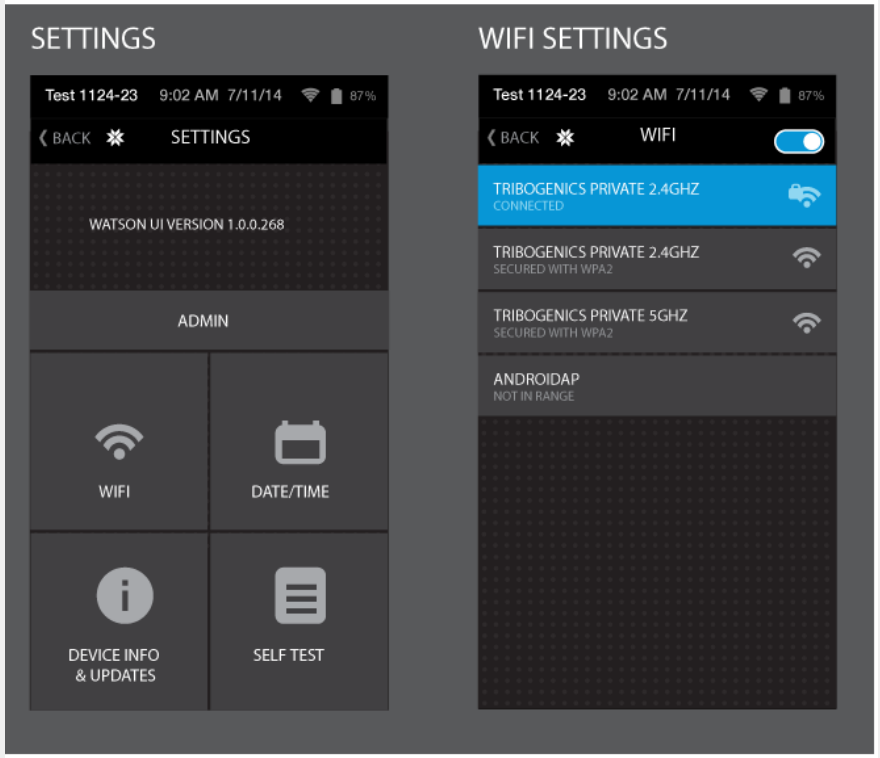
`frameworks/base/packages/SystemUI/src/com/android/systemui/statusbar/phone/PhoneStatusBar.java`

Altering the status bar - Change Icons

/frameworks/base/packages/SystemUI/src/com/android/systemui/statusbar/policy/NetworkController.java -- holds some code to remove cell signal icons:

```
...
else {
// normal mobile data
cluster.setMobileDataIndicators(
- mHasMobileDataFeature,
- mShowPhoneRSSIForData ? mPhoneSignalIconId : mDataSignalIconId,
+ false,
+ 0,
...
```

Reusing Android Settings



Reusing Android Settings

- Wi-Fi

Reusing Android Settings

- Wi-Fi
- Date / Time

Reusing Android Settings - Wi-Fi

SDG SYSTEMS™

packages/apps/Settings/src/com/android/settings/wifi/WifiSettings.java

```
onResume() {
...
activity.getActionBar().setDisplayHomeAsUpEnabled(true);
}

public boolean onOptionsItemSelected(MenuItem item) {
  if(item.getTitle().equals("Wi\u2011Fi")) {
    Log.d(TAG, "Hit the 'back' button, finishing the settings activity");
    finish();
    return true;
}
```

In your application:

```
Intent intent = new Intent(Settings.ACTION_WIFI_SETTINGS);
startActivity(intent);
```

Reusing Android Settings - Date / Time

packages/apps/Settings/src/com/android/settings/DateTimeSettings.java

```
onResume() {
...
activity.getActionBar().setDisplayHomeAsUpEnabled(true);
}

public boolean onOptionsItemSelected(MenuItem item) {
  if(item.getTitle().startsWith("Date")) {
    Log.d(TAG, "Hit the 'back' button, finishing the settings activity");
    finish();
    return true;
}
```

In your application:

```
Intent intent = new Intent(Settings.ACTION_DATE_SETTINGS);
startActivity(intent);
```

App-Specific Features

App-Specific Features

SDG SYSTEMS™

- USB hardware interaction

App-Specific Features

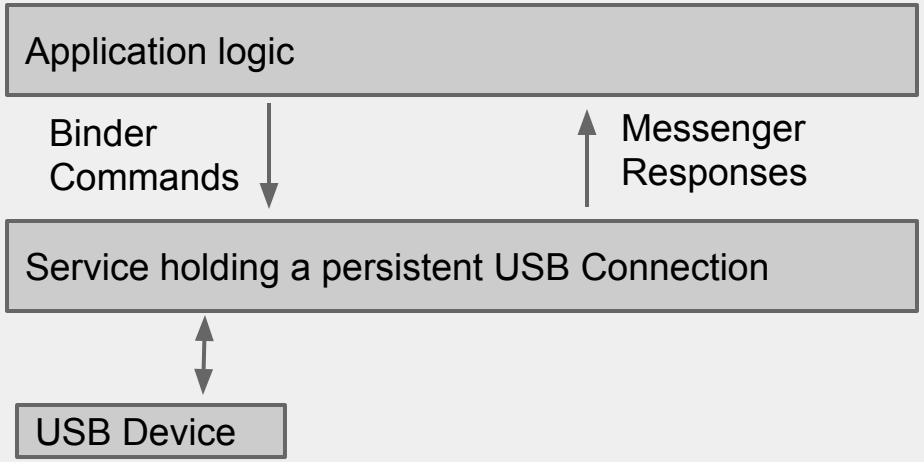
- USB hardware interaction
- Supporting an external video source w/ live preview and capture

App-Specific Features

- USB hardware interaction
- Remote Updates w/out the Google Play Store

USB hardware interaction

- USB Serial library
- Bound services
- Messengers as callbacks



USB hardware interaction - USB Serial library

https://github.com/mik3y/usb-serial-for-android

A FANTASTIC resource for anyone dealing with Android USB implementations.

USB hardware interaction - USB


```
class WatsonUsbDevice {
private UsbDeviceConnection usbConnection;
private android.hardware.usb.UsbDevice usbDevice;
private UsbSerialPort port;

public CustomUsbDevice(UsbDevice usbDevice, UsbDeviceConnection connection) {
  this.usbDevice = usbDevice;
  this.usbConnection = connection;
  UsbInterface intf = usbDevice.getInterface(0);
  usbConnection.claimInterface(intf, true);
  UsbSerialDriver driver = UsbSerialProber.getDefaultProber().probeDevice(usbDevice);
  for (UsbSerialPort port : driver.getPorts()) {
    this.port = port;
    break;
  }
  try { port.open(connection); }
  catch (Exception e) { log.error(Logs.DEV_DEBUG,"Exception in Device():", e); }
}
...
```

USB hardware interaction - Bound services

```
public class WatsonService extends Service {
...
     public class WatsonServiceBinder extends Binder {
       public WatsonServiceBinder() {
          super();
          binderInstance = this;
       }
       public WatsonService getService() {
          return WatsonService.this;
       }
       public void enableSource(boolean enable, int seconds) {
          if(watsonUsbDevice != null) {
               watsonUsbDevice.enableSourcePower(enable, seconds);
          }
       }
     }
}
```

USB hardware interaction - Bound services

```
private ServiceConnection mWatsonConnection = new ServiceConnection() {
     @Override
     public void onServiceConnected(ComponentName name, IBinder service) {
          mWatsonServiceBinder = (WatsonService.WatsonServiceBinder) service;
          watsonBound = true;
     }
     @Override
     public void onServiceDisconnected(ComponentName name) {
          mWatsonServiceBinder = null;
          watsonBound = false;
     }
};

Intent intent = new Intent(context, WatsonService.class);
mContext.bindService(intent, mWatsonConnection, Context.BIND_AUTO_CREATE);

mWatsonServiceBinder.enableSource(true, 10);
```

USB hardware interaction - Messengers


```
//Define the messenger and how to handle incoming messages
private Messenger mWatsonMessenger = new Messenger(new Handler() {
 @Override
 public void handleMessage(Message msg) {
  //filter out chatty battery messages
  if(msg.what != WatsonMessages.WATSON_MESSAGE_RESPONSE_BATTERY_GAUGE) {
       log.debug(Logs.EVENT, "got message " + msg.what);
  }
  //HANDLE THE MESSAGE based on msg.what
 }
}
```

USB hardware interaction - Messengers

```
private ServiceConnection mWatsonConnection = new ServiceConnection() {
     @Override
     public void onServiceConnected(ComponentName name, IBinder service) {
          mWatsonServiceBinder = (WatsonService.WatsonServiceBinder) service;
          mWatsonServiceBinder.registerMessenger(mWatsonMessenger);
          watsonBound = true;
     }
     @Override
     public void onServiceDisconnected(ComponentName name) {
          mWatsonServiceBinder.unregisterMessenger(mWatsonMessenger);
          mWatsonServiceBinder = null;
          watsonBound = false;
     }
};

Intent intent = new Intent(context, WatsonService.class);
mContext.bindService(intent, mWatsonConnection, Context.BIND_AUTO_CREATE);
```

USB hardware interaction - Messengers


```
WatsonService.java:
//In the service, register the messenger w/ the usb device
public void registerMessenger(Messenger messenger) {
     if(watsonUsbDevice != null) {
          watsonUsbDevice.registerMessenger(messenger);
     }
}

WatsonUSBDevice.java:
public void registerMessenger(Messenger clientMessenger) {
     if(this.mClientMessengers == null) {
          this.mClientMessengers = new ArrayList<Messenger>();
     }
     this.mClientMessengers.add(clientMessenger);
}
```

USB hardware interaction - Messengers


```
WatsonUSBDevice.java
Bundle dataBundle = new Bundle();
WatsonPacket packet;
if(bytes[1] == WatsonPacketTypes.COMMAND_RESPONSE_OPERATING_STATUS_HEADER[1]) {
     packet = new WatsonOperatingStatusPacket(bytes);
} else if (....

for (int i=mClientMessengers.size()-1; i>=0; i--) {
  Messenger mClientMessenger = mClientMessengers.get(i);
  Message dataMessage = packet.getMessageType();
  if(dataMessage != null) {
    dataBundle.putParcelable(WatsonMessages.KEY_DATA_BYTES, packet);
    dataMessage.setData(dataBundle);
    if(mClientMessenger != null) {
      mClientMessenger.send(dataMessage);
    }
  }
}
```

USB hardware interaction

For a more detailed example of services and messengers, check out our blog post:

http://sdgsystems.com/blog/using-android-ndk-android-studio-part-3/

Remote Updates w/out the Google Play Store

Remote Updates w/out the Google Play Store

- Update Service

Remote Updates w/out the Google Play Store

- Update Service
- Downloading / installing APKs programmatically

Remote Updates w/out the Google Play Store

SDG SYSTEMS™

- Update Service
- Downloading / installing APKs programmatically
- Implementing workflow deployment models

Remote Updates w/out the Google Play Store - Update Service

Remote Updates w/out the Google Play Store - Update Service

- Periodic background update service checking a stored URL

Remote Updates w/out the Google Play Store - Update Service

- Periodic background update service checking a stored URL
- Check / Verify current version

Remote Updates w/out the Google Play Store - Update Service

- Periodic background update service checking a stored URL
- Check / Verify current version
- Prompt user for manual updates

Remote Updates w/out the Google Play Store - Installing APKs

```
final UserManager um = (UserManager) _context.getSystemService(Context.USER_SERVICE);
Settings.Global.putInt(_context.getContentResolver(), Settings.Global.
INSTALL_NON_MARKET_APPS, 1);

//inside an asyncTask:
…
Intent intent = new Intent(Intent.ACTION_VIEW);

intent.setDataAndType(Uri.fromFile(new File("/mnt/sdcard/Download/watsonupdate.apk")),
"application/vnd.android.package-archive");

intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
_context.startActivity(intent);
```

Remote Updates w/out the Google Play Store - Deployment models

Using this update architecture, we were able to handle build deployment and promotion by targeting different update URLs.

Problems that we didn’t expect

SDG SYSTEMS™

Problems that we didn’t expect

- Debugging on an enclosed unit

Problems that we didn’t expect

- Debugging on an enclosed unit
- Android Studio support for statically linking pre-built NDK libraries

Problems that we didn’t expect

- Debugging on an enclosed unit
- Android Studio support for statically linking pre-built NDK libraries
- Handling USB charging while in host mode

Debugging an enclosed unit

SDG SYSTEMS™

Debugging an enclosed unit - ADB over Wi-Fi

The following will put the device's ADB connection in tcpip mode:

```
# adb tcpip 5555
restarting in TCP mode port: 5555
# adb connect 192.168.2.106
connected to 192.168.2.106:5555
```

Problem:
We need a USB connection to do set this up!

Debugging an enclosed unit - Setprop solution


```
on property:watson.adb.tcpip=1
    stop adbd
    setprop service.adb.tcp.port 5555
    start adbd

on property:watson.adb.tcpip=0
    stop adbd
    setprop service.adb.tcp.port -1
    start adbd
```

Without a rooted device, you cannot get user rights to issue setprop commands that init will be able to see.

Debugging an enclosed unit - Default properties

device/lge/hammerhead/aosp_hammerhead.mk:

```
PRODUCT_PROPERTY_OVERRIDES += \
    qemu.hw.mainkeys=1 \
    watson.adb.tcpip=1
```

Android Studio support for statically linking NDK libraries

SDG SYSTEMS™

Android Studio support for statically linking NDK libraries

PREBUILT_STATIC_LIBRARY not supported

Android Studio support for statically linking NDK libraries

build.gradle changes:

Android Studio support for statically linking NDK libraries

build.gradle changes:

- Override the default JNI compilation

Android Studio support for statically linking NDK libraries

build.gradle changes:

- Override the default JNI compilation
- Create a task to fire at compile time to manually compile JNI

Android Studio support for statically linking NDK libraries

build.gradle changes:

- Override the default JNI compilation
- Create a task to fire at compile time to manually compile JNI
- Add a properties file for ndk build path

Android Studio support for statically linking NDK libraries

```
apply from: '../properties.gradle'

defaultConfig {
 …
     sourceSets.main {
          jniLibs.srcDir 'src/main/libs'
          jni.srcDirs = [];
     }
}
…
tasks.withType(JavaCompile) {
     compileTask -> compileTask.dependsOn ndkBuild
}
task ndkBuild(type: Exec) {
     commandLine ndk_build_path, '-C', file('src/main/jni').absolutePath
}
```

Android Studio support for statically linking NDK libraries

properties.gradle:

```
project.ext.ndk_build_path = "/home/bfriedberg/android-ndk/android-ndk-r10/ndk-build"
```

src/main/jni/Android.mk:

```
…
include $(CLEAR_VARS)
LOCAL_MODULE    := libFP32
LOCAL_SRC_FILES := libs/$(TARGET_ARCH_ABI)/libFP32.a
include $(PREBUILT_STATIC_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE    := libMTFxpro
LOCAL_SRC_FILES := libs/$(TARGET_ARCH_ABI)/libMTFxpro.a
include $(PREBUILT_STATIC_LIBRARY)
…
```

Android Studio support for statically linking NDK libraries

For a more detailed example of linking in your ndk project:

http://sdgsystems.com/blog/using-android-ndk-android-studio-part-1/

Handling USB charging while in host mode

- Hostmode charging is prevented by default
- You won’t get notification
- OTG pin

How did you handle X?

Questions?